Resource Monitoring to Support Mobile Agents in Pervasive Computing

C. Brooks[1] and S. Krishnaswamy[2]
[1]School of Network Computing, Monash University, McMahons Road Frankston, Victoria 3199, Australia
E-mail: cbro7@student.monash.edu
[2]School of Computer Science and Software Engineering, Monash University
900 Dandenong Road, Caulfield East, Victoria 3145, Australia
E-mail: Shonali.Krishnaswamy@infotech.monash.edu.au

Abstract

Mobile agents are a form of software agents that migrate from device to device, performing their job in an unobtrusive manner. Mobile agents are particularly useful for pervasive computing as they are lightweight processes that allow for the delegation of tasks, have migratory characteristics, and operate regardless of whether the device they are on is connected to the network. However, in order to facilitate the full potential of mobile agents in pervasive environments, it is evident that there is a need for mobile agents to be resource-aware. To this end, we propose a model for a Resource Aware Middleware for Mobile Agents (RAMMA). We present the conceptual architecture of RAMMA, its implementation and an evaluation of the resource utilisation patterns of the monitor itself.

1. Introduction

Pervasive computing is the integration of mobile computing and ubiquitous computing; it is "anytime, anywhere" computing. The growth in pervasive computing relates to advances in portable device and networking technologies. Consequently, the number of vendors producing devices of this nature is continuously growing [1]. Many reasons have been given for this trend including improved wireless networks, decrease in device cost, increased worker productivity, and the need to provide mobile access to enterprise applications. As a result, companies such as Intel have noted that users are starting to perform job tasks from mobile rather than office locations [2]. This has also has led to increased worker productivity [3].

Software agents are autonomous software processes that perform tasks without the need for human interaction [4]. Mobile agents are the mobile form of software agents that migrate from device to device, performing their job in an unobtrusive manner [5]. Mobile agents are particularly useful for pervasive computing as they are lightweight processes that allow for the delegation of tasks, have migratory characteristics, and operate regardless of whether the device they are on is connected to the network. A network connection is only required when migrating [6].

In order that mobile agent applications can be developed and deployed, agent toolkits have been created. Many toolkits have been developed for creating agents, with several existing specifically for supporting agents on pervasive devices. These toolkits include: The Lightweight Extensible Agent Platform (LEAP) [7], Mobile Agents Environment (MAE) [8], Agent Factory [9], and Grasshopper (http://www.grasshopper.de).

At present, mobile agents arrive at their destination, perform their designed function, and leave. When this involves devices operating with constant power, relatively large amounts of memory, and a regular network connection, agents do not necessarily need to be aware of these particular resources. However, as previously indicated, the growth of pervasive computing involves the potential for agents to visit resource-limited devices. Considering

agents can arrive and execute on resource limited devices, remaining power, memory, and network availability become issues. For example, if an agent was to finish performing its task and try to leave when a network connection was not available, it must wait for one to become available. If one does not become available before battery power is depleted, and the agent does not have permission to write itself to persistent memory, it will be lost. Consider also the scenario where if memories were to be low, an agent may not be able to perform its task. Thus, it becomes evident that there is a need for mobile agents to be resource aware. To this end we propose a model for a Resource Aware Middleware for Mobile Agents (*RAMMA*).

Such a middleware as *RAMMA* should facilitate the operations of mobile agents in pervasive computing, and provide several functions and features to aid visiting mobile agents, including:

1. the ability to request the current available percentage of memory or power
2. the ability to discover whether the device is currently connected to the Internet
3. the ability to request connection reliability information
4. an alert system to inform agents when power and memory reserves are low.
5. be agent based so as to allow direct communication with agents
6. be well specified to ensure ease of use.

This paper begins with a discussion of related applications and architectures. It then proceeds to present a conceptual view of the proposed system, followed by an exploration of our implementation and evaluation of *RAMMA*. Finally, we provide our conclusions and future work intentions

2. Related Work

Operating systems provide resource information such as available memory, power, and network connection details upon request. Applications such as PowerScope [10, 11] provide information to the user about the supply and demand of energy for individual processes. There are other resource monitoring applications such as the work completed by Ranganathan et al [12] that provides mobile objects with resource information, and the work completed by Rocha and Toledo [13] that utilises resource monitoring to support transactions. However, these applications are not agent-based, and therefore, are not tailored for agent based interaction and communication. In CALMA [14] a framework whereby mobile agents are enabled with monitoring capabilities, is proposed. However, this places the onus of monitoring on individual agents and does not facilitate separation of concerns between resource monitoring and application function.

If a mobile agent were to posses the means to monitor resources, considering this information must be retrieved from the operating system, it will need to carry additional code. If the agent was to take into account the fact that it may visit any one of several individual operating systems, then it must equip itself with the knowledge to access this resource information on each system. This would potentionally increase the size of the agent to such a degree that it would no longer be a viable option to use a mobile agent [15]. Our approach is to provide an open, independent, agent-based, resource-monitoring service for visiting mobile agents.

3. *RAMMA* Architecture and Interactions

In this section we present the conceptual architecture for our Resource Aware Middleware for Mobile Agents. (Fig.1)

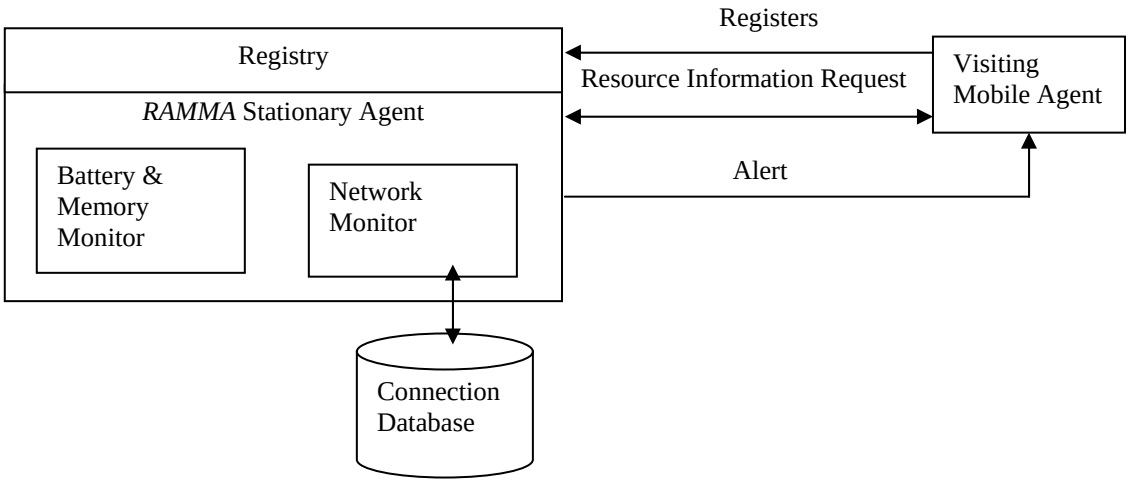

Fig. 1 Overview of *RAMMA* Architecture

Architecture

- **Visiting Mobile Agent**
  The *Visiting Mobile Agent* represents any agent that wishes to be provided with resource information. The agent can register for Alerts, or simply request resource information from the *RAMMA Stationary Agent*

- **Registry**
  When an agent arrives on a device, it can elect to register so as to receive Alerts when resources are low. During this registration the *Visiting Mobile Agent* must provide contact details in order for the *RAMMA Stationary Agent* to be able to contact it. These contact details are stored within the *Registry*.

- **RAMMA Stationary Agent**
  The *RAMMA Stationary Agent* runs continuously enabling *Visiting Mobile Agents* to register and request resource information at any stage.

  Battery and Memory Monitor
  The Battery and Memory monitor is responsible for all the power and memory resource related monitoring and queries. If a Visiting Mobile Agent requests some power or memory status information, it is the Battery and Memory monitor that handles this. Also when these resources become low the Battery and Memory monitor is responsible for informing the RAMMA Stationary Agent.

  Network Monitor
  The Network monitor is responsible for all the network related monitoring and queries. If a Visiting Mobile Agent requests some network status information, it is the Network monitor that handles this. Also the *Network Monitor* regularly queries the operating system to discover the current state of the network connection. Once a query has been performed, the result is stored in the Connection Database.

- **Connection Database**
  The Connection Database contains a history of the connection of this device to the Internet over a predefined period of time.

Interactions

A *Visiting Mobil Agent* registering with the *RAMMA Stationary Agent* is displayed in Fig. 2. Agents arriving on a device can elect to register at any stage. The details provided during this registration are used by the *RAMMA Stationary Agent* to provide Alerts when necessary

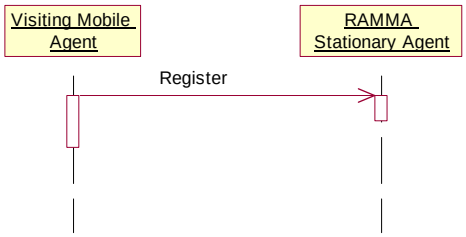

Fig. 2 Agent Registering

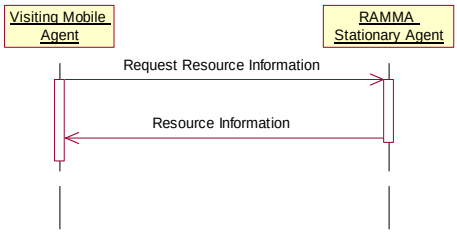

Fig. 3 Agent requesting resource information

A *Visiting Mobile Agent* requesting some resource information from the *RAMMA Stationary Agent* is illustrated in Fig 3. *Visiting Mobile Agents* can request the percentage of power and memory available on the device. They can also query The *RAMMA Stationary Agent* to discover information regarding the availability of a network connection, and how reliable the connection is based on how often this device has been within network range over a given period.

A registered *Visiting Mobil Agent* receiving an *Alert* is displayed in Fig 4. Once registered, *Visiting Mobile Agents* are informed via an Alert when resource levels are approaching, or have reached, critical levels. If an agent registers after a resource has gone below this level, it is immediately provided with an Alert to inform it of the current situation. In addition, registered agents will be notified via an Alert if the *RAMMA Stationary Agent* itself is being shut down

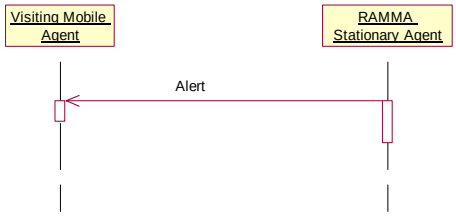

Figure 4 Agent receiving an Alert

Summary of Messages

Message Name	Message Description	Sender	Receiver
Register	An agent wishes to be informed when resources have reached predefined levels.	Visiting Mobile Agent	RAMMA Stationary Agent
Deregister	An agent no longer requires Alerts	Visiting Mobile Agent	RAMMA Stationary Agent

Request Power	A Visiting Mobile Agent wishes to know the current percentage of power remaining	Visiting Mobile Agent	RAMMA Stationary Agent
Request Memory	A Visiting Mobile Agent wishes to know the current percentage of memory remaining	Visiting Mobile Agent	RAMMA Stationary Agent
Request Connected	A Visiting Mobile Agent wishes to know whether this device is currently connected to the Internet	Visiting Mobile Agent	RAMMA Stationary Agent
Request Connection reliability	A Visiting Mobile Agent wishes to know the reliability of the Internet Connection	Visiting Mobile Agent	RAMMA Stationary Agent
Alert: Memory	The percentage of memory available has gone below a predefined level	RAMMA Stationary Agent	Visiting Mobile Agent
Alert: Power	The percentage of power available has gone below a predefined level	RAMMA Stationary Agent	Visiting Mobile Agent

Table 1 Summary of Messages between *RAMMA Stationary Agent* and *Visiting Mobile Agents*

The *RAMMA Stationary Agent* facilitates and aids *Visiting Mobile Agents* by giving them the ability to request the current available percentage of memory or power and connection reliability information. If agents register they are also provided with an alert system that informs them when power and memory reserves are low. *Visiting Mobile Agents* can also discover if the device they are on is currently connected to the Internet. In addition, the interactions are well specified through the use of Alerts.

In the following sections we discuss our implementation of *RAMMA* and provide performance evaluation based on the fact that *RAMMA* itself consumes resources.

4. Implementation

In this section we describe our implementation of *RAMMA* and present two scenarios to illustrate its abilities to facilitate and aid the operating of mobile agents in pervasive computing.

In order to create an implementation of *RAMMA*, we chose the GrassHopper agent toolkit due to its support for the Windows CE operating system. To enable *RAMMA* to communicate with the Windows CE operating system a dynamic link library must be registered. *RAMMA* is created as a stationary agent so as to enable communication. Visiting mobile agents would have difficulty discovering *RAMMA* if it were operating as a stand-alone middleware. However, when implemented as a stationary agent, the environment (toolkit) has the facilities to enable discovery and bidirectional communication. This is particularly relevant when considering a resource limited device, as discovery of the middleware may consume valuable resources, and require the agent to have knowledge of the operating system.

In addition, visiting agents must know how to interpret an "Alert" that have been generated by *RAMMA* and be aware of the class description, as shown in Figs. 5 & 6. In order to receive "Alerts" visiting agents must also implement the IAlertReceiver interface.

Constructor Summary
Alert(int severity, int type, boolean connected)
Method Summary

Boolean **getConnected**() Returns true if an Internet connection is currently available.
int **getSeverity**() Returns 0 for critical, 1 for very low, 2 for low, 3 for almost low, and 4 *RAMMA* is shutting down;
int **getType()** Returns 0 for a memory Alert, 1 for a power Alert, and 2 for *RAMMA* shutting down

Fig. 5 Alert class

```
Public interface IAlertReceiver
{
    public void alert (Alert alert);
}
```

Fig. 6 IAlertReciever Interface

Scenario 1

As illustrated in Fig. 7, an agent arrives on the device and registers with *RAMMA*. The agent wants to conduct some communication with a web service and this communication will require a constant connection for a certain period of time. The agent does not want to repeat this communication because it has been interrupted, so it wishes to know how reliable the current connection is. To this end it requests the available power and memory from *RAMMA* to discover if the device's resources will last through the communication. Discovering that the resources levels are acceptable, it makes a request to *RAMMA* and is provided with an indication of this device's current connection reliability. This is followed by a call to *RAMMA* in order to discover if the device is currently connected to the Internet. Figure 8 displays the potential problem for this agent if it cannot be provided with the relevant information, and the network is unreliable.

Fig. 7 Agent utilising RAMMA *Stationary Agent*

Fig. 8 *RAMMA* not running and no network available, agent failing to communicate

Scenario 2

A registered agent is performing a task, when an alert arrives (Fig 8). The alert is the initial power alert, so the agent chooses to ignore it. The agent is performing an important task and will only be interested if the alert is critical. Once the critical power alert has been received, the agent deregisters and leaves.

If the agent was to arrive and begin execution, considering it is performing an important task, it may not complete before the power is depleted. If *RAMMA* was not available to this agent, and the power resource was exhausted, the agent would be destroyed. For obvious reasons, no illustration is provided to display this.

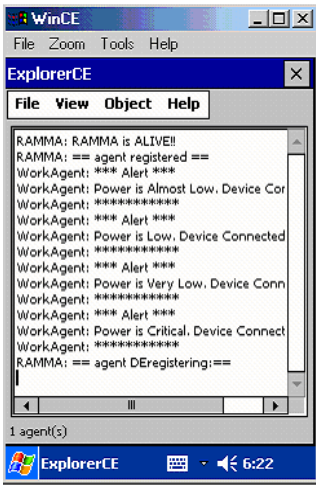

Fig. 8 Agent receiving Alerts

The scenarios above illustrates that the implementation meets all the design criteria of functionality specified in the introduction. Scenario one shows a V*isiting Mobile Agent* requesting power and memory resource information. It also demonstrates *RAMMA* providing connection reliability and current Internet connectivity information. Scenario two displays an agent receiving Alerts since the power has gone below pre-defined levels. Both scenarios demonstrate an agent registering so as to be supplied with Alerts if the system resources are below pre-defined levels; scenario two demonstrates an agent that is deregistering.

5. Evaluation

In this section we evaluate the resource consumption patterns of *RAMMA*. This evaluation is performed to discover the resource cost associated with running *RAMMA*, as it also consumes resources.

For the evaluation of *RAMMA*, an IPAQ 3970, with and Intel PXA250 (arm) processor, 31.56 MB of program memory, Li-ion battery, and Microsoft Pocket PC 3.0.1 were used. All tests were run over a period of 5 consecutive hours, with readings taken every half an hour.

5.1 Analysis of RAMMA Power consumption

Experiment

In order to discover the extent of power consumption by *RAMMA*, we must first measure the difference between the power consumption of the operating system, and the components that are required by *RAMMA*. We then measure the power consumption of *RAMMA* with varying connectivity levels to more accurately illustrate this consumption.

Aim

RAMMA requires an operating system, a virtual machine and an agent toolkit in order to be run. The aim of the experiment is to show the power consumption of these required applications and to compare this to the overall consumption of *RAMMA*

Case 1:

Case one considers the power consumption of *RAMMA* and supporting applications.

Results

A comparison of power consumption of just the operating system, the operating system with a virtual machine and GrassHopper, and the operating system, virtual machine, GrassHopper and *RAMMA*, is displayed in Fig. 9. On average the operating system consumes approximately 2.7% of the total power every half hour, the operating system, virtual machine and GrassHopper consume approximately 2.8%, and the operating system, virtual machine, GrassHopper and *RAMMA* consume approximately 2.9% of the total power every half hour.

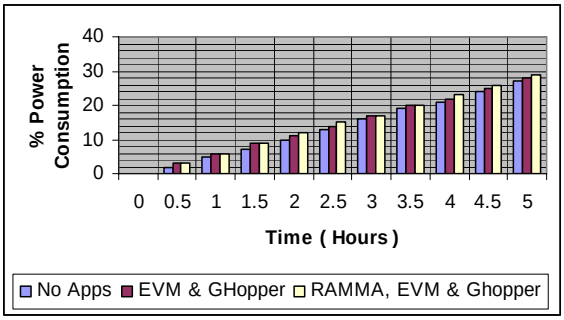

Fig. 9 Comparison of Power Consumption

Analysis for Case 1:

The test performed illustrates that *RAMMA* consumes approximately 0.1% of the total power every half hour. Considering that *RAMMA* requires a virtual machine and agent toolkit, the results reveal that *RAMMA* consumes approximately 0.2% of the total power every half hour.

Case 2:

Case two considers the power consumption of *RAMMA* when there are varying connectivity levels.

Results

Fig. 10 illustrates the power consumption of *RAMMA* when running with no network capabilities, of *RAMMA* when running with network capabilities, although disconnected, and of *RAMMA* when running with network capabilities and with a current connection to the Internet. On average, *RAMMA* consumes approximately 2.9% of the total power every half-hour. When the network is enabled and no connection exists, RAMMA consumes approximately 3.0% of the total power, and when RAMMA has a current connection to the Internet, it consumes approximately 2.8% of the total power every half-hour.

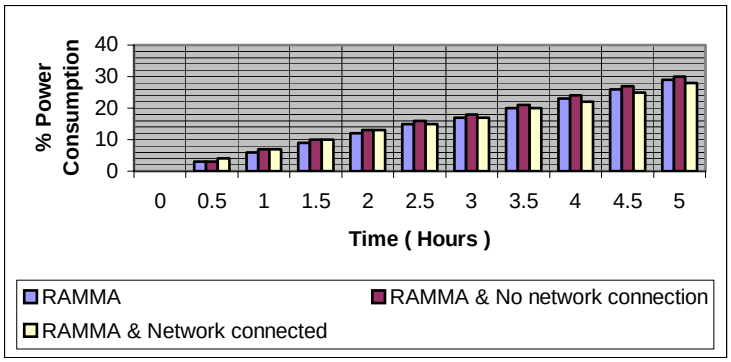

Fig 10 Stand alone comparison of Power Consumption

Analysis for Case 2:

The test performed illustrate that the total power consumption varies between approximately 2.8% and 3.0% of the total power every half hour when *RAMMA* is running.

Analysis: Power consumption *RAMMA*

The tests performed demonstrate that the power consumption of *RAMMA*, provided it has network capabilities available, varies between approximately 2.8% and 3.0% every half hour. When the power consumption of the operating system is taken into account, *RAMMA* consumes between 0.1% and 0.3% of the total power every half hour. The results also suggest that power consumption is reduced when a network connection is available.

5.2 Analysis of RAMMA memory usage

Experiment

In order to discover the extent *RAMMA* utilises memory, we must first measure the difference between the utilisation of the operating system, and the components that are required by *RAMMA*. We then measure the memory utilisation of *RAMMA* with varying connectivity levels to more accurately illustrate this.

Aim

RAMMA requires an operating system, a virtual machine and an agent toolkit in order to be run. The aim of the experiment is to show the memory utilisation of these required applications and to compare this to the overall usage of *RAMMA*

Case 1:

Case one considers the memory usage of *RAMMA* and supporting applications.

Results

A comparison of the memory usage of just the operating system, the operating system with a virtual machine and GrassHopper, and the operating system, virtual machine, GrassHopper and *RAMMA*, is shown in Fig. 11. On average the operating system utilises approximately

21% of the total memory, the operating system, virtual machine and agent toolkit utilises approximately 29%, and the operating system, virtual machine, agent toolkit and *RAMMA* consume approximately 31% of the total memory.

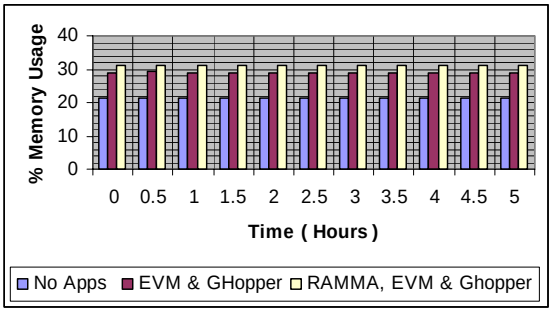

Fig. 11 Comparison of Memory usage

Analysis for Case 1:

The tests performed illustrate that whilst *RAMMA* is running, 31% of the total memory is being utilised. Considering that *RAMMA* requires a virtual machine and agent toolkit, the results reveal that *RAMMA* uses on average approximately 10% of the total memory, as the operating system utilises 21% .

Case 2:

Case two considers the memory usage of *RAMMA* when there are varying connectivity levels.

Results

Displayed in Fig 12 is the memory usage of *RAMMA* when running with no network capabilities, of *RAMMA* when running with network capabilities, although disconnected, and of *RAMMA* when running with network capabilities and with a current connection to the Internet. On average *RAMMA* utilises approximately 31% of the total memory, when network enabled and no connection RAMMA utilises approximately 34% of the total memory, and when RAMMA has a current connection to the Internet, it utilises approximately 34% of the of the total memory.

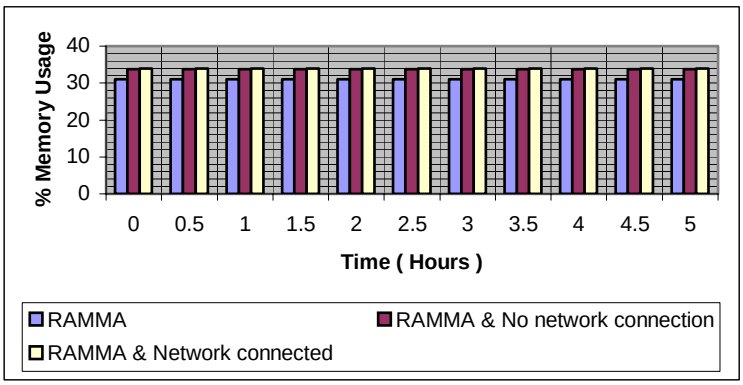

Figure 12 Stand alone comparison of Memory usage

*Analysis for Case 2***:**

The tests performed illustrate that the total memory utilization varies between approximately 31% and 34% when *RAMMA* is running. When the memory usage of the operating system is taken into account, *RAMMA* utilises on average 10% - 13% of the total available memory.

Analysis: Memory Usage *RAMMA*

The tests performed demonstrate that the memory utilisation of *RAMMA*, provided it has network capabilities available, is on average approximately 34%. However, when the memory utilisation of the operating system is taken into account, the utilisation of *RAMMA* only represents 10% - 13%. The fluctuation of 3% represented can be attributed to network communication

6. Conclusion & Future Work

Pervasive computing is increasing as advances in portable device and networking technologies continue. This, in turn, has led to the increased potential for agents to visit resource-limited devices. *RAMMA* provides the means by supplying an architecture that removes the need for agents to provide their own method of resource awareness. Our evaluation of an implementation adhering to this architecture has revealed relatively acceptable amounts of resource use and consumption. Thus, we conclude that the potential benefits of allowing agents the greatest possibility of leaving a device before resource levels become a concern, outweigh the cost of providing this valuable service.

At present, our implementation can only be utilised by GrassHopper agents and on the Windows CE operating system. In order to meet this limitation our future work will involve converting the implementation to a Knowledge Query and Manipulation Language (KQML) [16] compliant application, to provide mobile agents, developed using other toolkits, the ability to communicate with *RAMMA Stationary Agent*.

References

[1] MarketWatch: wireless, Enterprise mobile device market analysis. (Industry Overview), *MarketWatch: wireless*, May 2003.

[2] K. Shaw, Wireless workers travel light, work smarter, study finds, *Network World Wireless Computing Devices Newsletter*, November 2003.

[3] K. Shaw, How wireless changed the work behavior of Intel employees, *Network World Wireless Computing Devices Newsletter*, September 2003.

[4] H.S. Nwana, Software Agents: An Overview, *Knowledge Engineering Review*, Vol. 11, No 3, pp.205-244, October/November 1996.

[5] S. Fünfrocken & F. Mattern, *Mobile Agents as an Architectural Concept for Internet-based Distributed Applications: The WASP Project Approach.* In: Steinmetz (Ed.): Proc. KiVS'99, pp. 32-43, Springer-Verlag, 1999

[6] L.B. Danny, & M. Oshima, Seven good reasons for mobile agents. *Communications of the ACM*, March 1999. Vol. 42, Issue 3, 88 – 89, March 1999

[7] F. Bergenti & A. Poggi, LEAP: A FIPA platform for handheld and mobile devices. In *Proceedings of the Eight International Workshop on Intelligent Agents ATAL 2001*, August 2001.

[8] P. Mihailescu & E. A. Kendall., Development of an agent platform for mobile devices using J2ME. In *Proceedings of the Evolve 2001 conference*, Sydney, Australia, May 2001.

[9] R.W. Collier & G.M.P. O'Hare, , Agent Factory: A Revised Agent Prototyping Environment, *10th AICS Conference, Irish Artificial Intelligence and Cognative Science Conference*, Cork, Ireland, 1999.

[10] J. Flinn & M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Proceedings of the Symposium on Operating Systems Principles*, pages 48-63, 1999.

[11] J. Flinn & M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications WMCSA'99*, Feb 1999.

[12] M. Ranganathan, A. Acharya, & J. Saltz, Distributed Resource Monitors for Mobile Objects In *Proceedings of the Fifth International Workshop on Object Orientation in Operating Systems*, pages 19-23, Seattle, Wa, October 1996.

[13] T. Rocha & M.B.F. Toledo, (2003) "A System of Adaptable Transactions for The Mobile Computing Environment" <Available online (Accessed 7th March 2004)> http://middleware2003.inf.puc-rio.br/posters/rocha.pdf

[14] A. Sumartono, (2003), "A Light-weight Mobile BDI Agent Approach for Context-Aware Execution of Web Services", Masters of Information Technology Thesis. Monash University Australia.

[15] M. Straßer and M. Schwehm, A Performance Model for Mobile Agent Systems, in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications* (PDPTA'97), (eds) H. Arabnia, Vol II, CSREA, 1132-1140, 1997

[16] T. Finin, Y. Labrou & Mayfield, J., KQML as an Agent Communication Language, *Proceedings of the Third International Conference on Information and Knowledge Management* (CIKM'94), ACM Press, November 1994.